ARMlite Experimental and Advanced Features

These features were added to ARMlite to enable more advanced programs to be written. Richard and Peter decided to leave them out of the Programming Reference Manual (and the Book) because they are not needed at this stage and in the last case we might do a different, industry standard, floating point if we ever did a more advanced textbook.

Numeric Indexing

The Book and the Programming Reference Manual introduce indirect [Rn] and indexed [Rn+/-Rm] addressing and there is a side reference in the Manual to [PC+/-n] as a way of getting 'pseudo direct addresses'.

In fact ARMlite supports [Rn+n] and [Rn-n] as relative or offset addressing with any register. As with indirect and indexed, these modes can only be used with LDR, LDRB, STR and STRB to move a value between a register and memory. (E.g. STR R4,[R5+0x44].) n can range from 0 to 4095 and can be given in binary, decimal or hexadecimal and the ARM formats are supported (e.g. STR R4,[R50,#68] but not the scaling or write back capabilities).

Typical uses of this mode might be to implement a 'struct' where the register contains either the base of the struct or the base of a data area. Also it is conventional to stack register parameters on entry to a subroutine or function and to allocate local variables on the stack. These can then be accessed as [SP+n] or more normally (for example) as [R12+n] for parameters and [R12-n] for locally allocated variables.

Because ARMlite programs are often small, where a label is less than 4095, ARMlite allows [Rn+label] and the Assembler converts the label into a number. This particular case is not of general use on a real ARM processor.

ARM Bit Shift Instructions

Release v1.2 of ARMlite supports all the basic ARM bit shifting instructions. Note these are only supported as stand alone instructions and not in combination with other instructions. LSR and LSL are part of the basic AQA set and still supported. ASL (Arithmetic Shift Left) is now accepted as a synonym for LSL and the following are newly supported.

ASR

'Arithmetic Shift Right'. Moves each bit to the right, duplicating the left-most bit (i.e. the sign bit) and losing the right-most bit off the end. If the value represents an signed number, the effect of shifting one place right is to divide the number by two, rounded down if appropriate.

ROR

'ROtate Right'. Moves each bit to the right, using the right-most bit as the new value for the left-most bit.

RRX

'Rotate Right with eXtend'. Moves each bit one place to the right, using the C flag as the new value for the left-most bit. (Note you need RRXS to move the right most bit to the C flag as well.)

In addition the 'Set flags' versions of all the above are supported. I.e LSRS, LSLS, ASLS, ASRS, RORS and RRXS. The V flag is unchanged, the C flag is set to the last lost bit and the N (negative) and Z (zero) flags are set according to the final result. (Note for non-S versions the flags are unchanged.)

For all the shift instructions apart from RRX (which is one place only) a variable number of shifts can be made. For each of the following examples, `R0` starts with `0b1110` and `R1` starts with `0b1011`, and all values not shown to 32 bits have all leading zeros.

`ASL R2, R0, #2`	results in `R2` containing `0b111000` (the value of `R0` shifted left 2 places)
`ASR R2, R0, #1`	results in `R2` containing `0b111`
`ROR R2, R0, #3`	results in `R2` containing `0b1100000000000001`
`RRX R2, R0`	results in `R2` containing `0b1000000000000111` if C is 1
`ASL R2, R0, R1`	results in `R2` containing `0b111000000000000`

If the shift count is immediate (#n) then n can be in range 1-31 and 32 is allowed for ASR and LSR. #0 is never allowed and the assembler will error. (Note LSL with #0 is the encoding for MOV.)

If the shift count is the contents of a register then only the bottom 8 bits are used but (within those 8 bits) any value over 32 is taken as 32. (There is no variant of RRX that takes a count.)

Floating Point Input and Output

ARMlite additionally supports .InputFP and .WriteFP which are used in the same way as the other numeric I/O addresses to input and output floating point numbers in the AQA Floating Point representation format, where negative values for the mantissa and/or exponent are represented in two's complement.

ARMlite allocates the top 24 bits to the mantissa, and the least significant 8 bits to the exponent. This gives a maximum range of about +/- 10^{38} and an accuracy of about 7 significant decimal digits.

ARMlite has no instructions to manipulate Floating Point. We envisioned the possibility of student exercises such as write programs to add or multiply floating point numbers in this simplified format. To make this possible, ARMlite provides functions to input and output floating point values in this format.

This is not in the standard Manual because we want to retain the option to provide industry standard floating point in a future version.